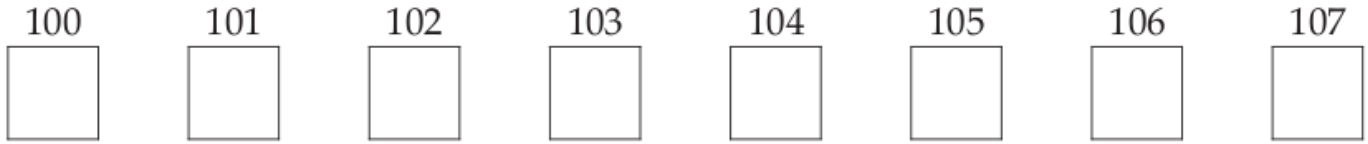
Memoria y direcciones

La memoria de las computadoras están compuestas de millones de `bits`, cada uno capaz de tener el valor de 0 o de 1 agrupados como una unidad para almacenar un rango mayor de valores.



1. Cada localización es llamado un `byte`.
2. Cada localización en memoria es identificado por una única dirección.
3. Cada localización en memoria contiene un valor.

El siguiente gráfico muestra 5 valores enteros, cada uno ocupando una cantidad de `bytes`.

100	104	108	112	116
112	-1	1078523331	100	108

• Los lenguajes de programación de alto nivel proporcionan la habilidad de referirse a la localización de memoria por nombres que por direcciones y esos nombres son llamados `variables`.

a	b	c	d	e
112	-1	1078523331	100	108

Veamos las declaraciones de este gráfico

```
int a= 112, b = -1;
float c = 3.14;
int *d = &a;
float *e = &c;
```

Las variables `d` y `e` fueron declaradas como punteros y son inicialiazadas con la dirección de otra variable. La inicialización es hecha con el operador &, que produce la dirección de memoria de su operando.

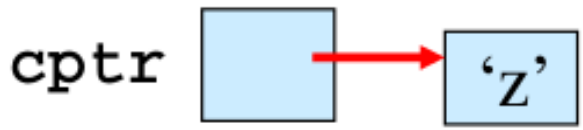
Punteros en C

1. Una variable puntero almacena la dirección de una localización de memoria que almacena el tipo al cual apunta.

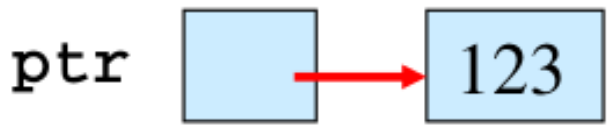
```
int *ptr   // almacena la direccion de un entero.
           // ptr 'apunta a' un int.

char *cptr // almacena la direccion de un char.
           // cptr 'apunta a' un char.
```

2. El tipo de `cptr` es un puntero a char. Este apunta a una localización de memoria que almacena un valor char. Mediante `cptr` podemos acceder indirectamente un valor char.



3. El tipo de `ptr` es un puntero a un entero. Este apunta a una localización de memoria que almacena un valor int.

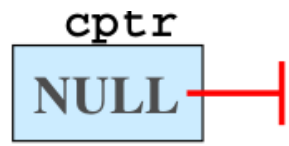


Inicializando variables puntero

- Como una variable, se debe inicializar el puntero antes de que se pueda utilizar.
- Asigna la variable puntero el valor de una dirección de memoria que pueda almacenar el tipo al cual apunta.

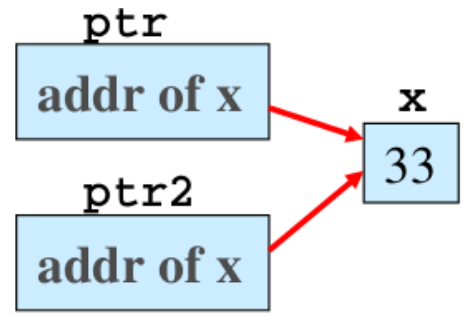
1. NULL es un especial valor para punteros. No es una dirección válida.

```
char *cptr = NULL;
```



2. El operador & evalua la dirección de una variable argumento.

```
int x = 3;
int *ptr = NULL, *ptr2 = NULL;
ptr = &x; // Consigue la direccion de x
          // ptr 'apunta a' x
ptr2 = ptr; //ptr2 obtiene el valor de ptr
  // ambos apunta a la misma localizacion
char *cptr = &x; // ERROR!?.
```



El uso de punteros

Una vez que un puntero es inicializado apuntando a una localización de almacenamiento válido, se puede acceder al valor al que apunta usando el operador *

```
* : dereferenciando una variable puntero
(accede a la localizacion de almacenamiento a la que apunta.)
```

```
ptr = &x; // ptr tiene la direccion de x 'ptr apunta a x'
*ptr = 10  // almacena 10 en la localizacion que ptr apunta
```


```
cptr = NULL;
*cptr = 'b';    // cptr no apunta a una localizacion de almacenamiento
                // char valida, tratando de dereferenciar cptr
                // (un puntero NULL) se colgara el programa

if(cptr != NULL){ // Una mejor manera es probar para  NULL primero.
*cptr = 'b';  // El ajuste del puntero NULL, permite probar
              // por un direccion invalida antes de la  desreferenciacion.
}
```